Reliable CRC-Based Error Detection Constructions for Finite Field Multipliers With Applications in Cryptography

Alvaro Cintas Canto, Mehran Mozaffari-Kermani, and Reza Azarderakhsh

***Abstract*— Finite-field multiplication has received prominent attention in the literature with applications in cryptography and error-detecting codes. For many cryptographic algorithms, this arithmetic operation is a complex, costly, and time-consuming task that may require millions of gates. In this work, we propose efficient hardware architectures based on cyclic redundancy check (CRC) as error-detection schemes for postquantum cryptography (PQC) with case studies for the Luov cryptographic algorithm. Luov was submitted for the National Institute of Standards and Technology (NIST) PQC standardization competition and was advanced to the second round. The CRC polynomials selected are in-line with the required error-detection capabilities and with the field sizes as well. We have developed verification codes through which software implementations of the proposed schemes are performed to verify the derivations of the formulations. Additionally, hardware implementations of the original multipliers with the proposed error-detection schemes are performed over a Xilinx field-programmable gate array (FPGA), verifying that the proposed schemes achieve high error coverage with acceptable overhead.**

***Index Terms*— Cyclic redundancy check (CRC), fault detection, field-programmable gate array (FPGA), finite-field multiplication.**

I. Introduction

Many modern, sensitive applications and systems use finite-field operations in their schemes, among which finite-field multiplication has received prominent attention. Finite-field multipliers perform multiplication modulo, an irreducible polynomial used to define the finite field. For postquantum cryptography (PQC), the inputs can be very large, and the finite-field multipliers may require millions of logic gates. Therefore, it is a complex task to implement such architectures resilient to natural and malicious faults; consequently, research has focused on ways to eliminate errors and obtain more reliability with acceptable overhead [1]–[6]. Moreover, there has been previous work on countering fault attacks and providing reliability for PQC. Sarker *et al.* [7] used error-detection schemes of number theoretic transform (NTT) to detect both permanent and transient faults. Mozaffari-Kermani *et al.* [8] performed fault detection for stateless hash-based PQC signatures. Additionally, error-detection hash trees for stateless hash-based signatures are proposed in [9] to make such schemes more reliable against natural faults and help protecting them against malicious faults. In [10], algorithm-oblivious constructions are proposed through recomputing with swapped ciphertext and additional authenticated blocks, which can be applied to the Galois counter mode (GCM) architectures using different finite-field multipliers in $GF(2^{128})$. Several countermeasures based on error-detection checksum codes and spatial/temporal redundancies for the NTRU encryption algorithm have been presented in [11].

Our proposed error-detection architectures are adapted to the Luov cryptographic algorithm [12]; however, they can be applied to different PQC algorithms that use finite-field multipliers. The Luov algorithm was submitted for National Institute of Standards and Technology (NIST) standardization competition [13] and was advanced to the second round [14]. Cyclic redundancy check (CRC) error-detection schemes are applied in our proposed hardware constructions to make sure that they are overhead-aware with high error coverage. Our contributions in this brief are summarized as follows.

1) Error-detection schemes for the finite-field multipliers $GF(2^m)$ with $m > 1$ used in the Luov cryptographic algorithm are proposed. These error-detection architectures are based on CRC-5. Additionally, we explore and study both primitive and standardized generator polynomials for CRC-5, comparing their complexity.
2) We derive new formulations for the error-detection schemes of Luov's algorithm, performing software implementations for the sake of verifications. We note that such derivation covers a wide range of applications and security levels. Nevertheless, the presented schemes are not confined to these case studies.
3) The proposed error-detection architectures are embedded into the original finite-field multipliers. We perform the implementations using Xilinx field-programmable gate array (FPGA) family Kintex Ultrascale+ for device xcku5p-ffvd900-1-i to confirm that the schemes are overhead-aware and that they provide high error coverage.

II. Preliminaries

There are five popular PQC algorithm classes: code-based, hash-based, isogeny-based, lattice-based, and multivariate-quadratic-equation-based cryptosystems [15]. Code-based cryptography differs from others in that its security relies on the hardness of decoding in a linear error-correcting code. Hash-based cryptography creates signature algorithms based on the security of a selected cryptographic hash function. The security of isogeny-based cryptography is based on the hard problem to find an isogeny between two given supersingular elliptic curves. Lattice-based cryptography is capable of creating a public-key cryptosystem based on lattices. Lastly, the security of multivariate-quadratic-equation-based cryptography depends on the difficulty of solving a system of multivariate polynomials over a finite field. Such cryptographic schemes use large field sizes to provide the needed security levels.

Luov is a multivariate public key cryptosystem and an adaptation of the unbalanced oil and vinegar (UOV) signature scheme, but there is a restriction on the coefficients of the public key. Instead, the scheme uses two finite fields: one is the binary field of two elements, whereas the other is its extension of degree m. F_2 is the binary field and F_{2^m} is its extension of degree m. The central map $F\colon F_{2^m}^n \rightarrow F_{2^m}^o$ is a quadratic map, where o and v satisfy $n = o + v$, $\alpha_{i,j,k}$, $\beta_{i,k}$ and γ_k

Manuscript received May 13, 2020; revised August 8, 2020 and September 18, 2020; accepted October 11, 2020. Date of publication October 26, 2020; date of current version December 29, 2020. This work was supported by the U.S. National Science Foundation (NSF) under Award SaTC-1801488. *(Corresponding author: Mehran Mozaffari-Kermani.)*

Alvaro Cintas Canto and Mehran Mozaffari-Kermani are with the Department of Computer Science and Engineering, University of South Florida, Tampa, FL 33620 USA (e-mail: alvarocintas@usf.edu; mehran2@usf.edu).

Reza Azarderakhsh is with the Department of Computer and Electrical Engineering and Computer Science and I-SENSE, Florida Atlantic University, Boca Raton, FL 33431 USA (e-mail: razarderakhsh@fau.edu).

Digital Object Identifier 10.1109/TVLSI.2020.3031170

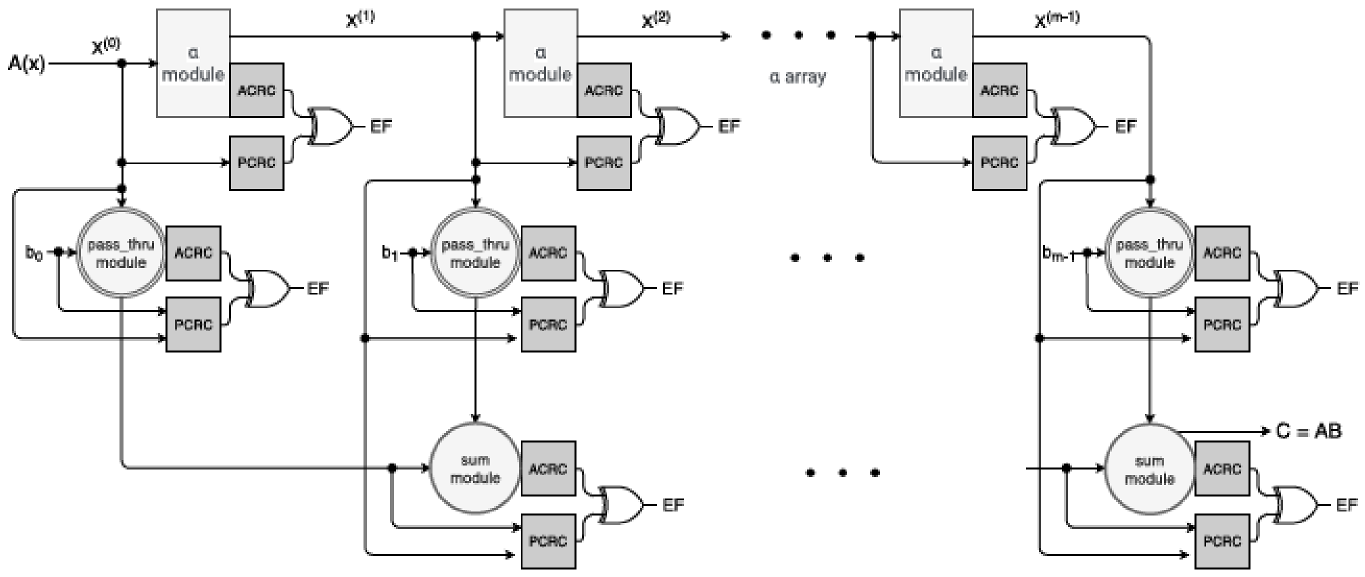

Fig. 1. Finite-field multiplier with the proposed error-detection schemes based on CRC.

are chosen from the base field F_2, and whose components $f_1, \ldots, f_o$ are in the form $fk(x) = \sum_{i=1}^{v}\sum_{j=i}^{n} \alpha_{i,j,k}x_ix_j + \sum_{i=1}^{n}\beta_{i,k}x_i + \gamma_k$.

These finite-field multiplications are very complex and require large-area footprint. Therefore, it is a complex task to implement such architectures resilient to natural and malicious faults. The aim of this work is to provide countermeasures against natural faults and fault injections for the finite-field multipliers used in cryptosystems such as the Luov algorithm as a case study, noting that the proposed error-detection schemes can be adapted to other applications and cryptographic algorithms whose building blocks need finite-field multiplications. Readers who are interested in knowing more details about the Luov's cryptographic algorithm are encouraged to refer to [12].

III. Proposed Fault-Detection Architectures

The multiplication of any two elements A and B of $GF(2^m)$, following the approach in [16], can be presented as $A \cdot B \bmod f(x) = \sum_{i=0}^{m-1} b_i \cdot ((A\alpha^i) \bmod f(x)) = \sum_{i=0}^{m-1} b_i \cdot X^{(i)}$, where the set of α^i's is the polynomial basis of element A, the set of b_i's is the B coefficients, $f(x)$ is the field polynomial, $X^{(i)} = \alpha \cdot X^{(i-1)} \bmod f(x)$, and $X^{(0)} = A$. To perform finite-field multiplication, three different modules are needed: *sum*, α, and *pass-thru* modules. The *sum* module adds two elements in $GF(2^m)$ using m two-input XOR gates, the α module multiplies an element of $GF(2^m)$ by α and then reduces the result modulo $f(x)$, and lastly, the *pass-thru* module multiplies a $GF(2^m)$ element by a $GF(2)$ element. One finite-field multiplication uses a total of $m-1$ *sum* modules, $m-1$ α modules, and m pass-thru modules to get the output. Fault injection can occur in any of these modules, and formulations for parity signatures in $GF(2^m)$ are derived in [16]. Parity signatures provide an error flag (EF) on each module. The major drawback of parity signatures is that their error coverage is approximately 50%, that is, if the number of faults is even, the approach would not be able to detect the faults. This highly predictable countermeasure can be circumvented by intelligent fault injection.

In this work, our aim is the derivation of error-detection schemes that provide a broader and higher error coverage than parity signatures and explore the application of such schemes to the Luov algorithm. Thus, we derive and apply CRC signatures [17] to the finite-field multipliers used in Luov algorithm. This would be a step forward toward detecting natural and malicious intelligent faults, especially and as discussed in this brief, considering both primitive and standardized CRCs with different fault multiplicity coverage. CRC was first proposed in 1961 and it is based on the theory of cyclic error-correcting codes. To implement CRC, a generator polynomial $g(x)$ is required. The message becomes as the dividend, the quotient is discarded, and the remainder produces the result. In CRC, a fixed number of check bits are appended to the data and these check bits are inspected when the output is received to detect any errors.

The entire finite-field multiplier with our error-detection schemes is shown in Fig. 1, where actual CRC (ACRC) and predicted CRC (PCRC) stand for ACRC signatures and PCRC signatures, respectively. In Fig. 1, only one EF is shown for clarity; however, for CRC-5, which is the case study proposed in this brief, 5 EFs are computed on each module. In Fig. 2, the α module is shown more in-depth to clarify how the proposed CRC signatures work in each finite-field multiplier.

For the *sum* and *pass-thru* modules, it follows the approach as for parity signatures described in [16]. For the *sum* module in CRC-1, $\hat{p}_x$ is equal to the sum of the parity bits of the input elements A and B in $GF(2^m)$, $\hat{p}_X = p_A + p_B$. Furthermore, for the *pass-thru* module in CRC-1, $\hat{p}_X = b \cdot p_A$, where b is an element in $GF(2)$. For any other CRC-n scheme, instead of summing all the bits, it checks n bits at a time in the *sum* and *pass-thru* modules. For the α module, we have

$$A(x) \cdot x = a_{m-1} \cdot x^m + a_{m-2} \cdot x^{m-1} + \cdots + a_0 \cdot x \qquad (1)$$

for which a set of derivations is needed to implement CRC-n into it. In Table I, the generator polynomials used to derive the CRC-5 signatures are shown. The generator polynomial $g_0(x)$ is one of the standards used for radio frequency identification [18]. The other three generator polynomials $g_1(x)$, $g_2(x)$, and $g_3(x)$ are primitive polynomials. The benefit of using a primitive polynomial as the generator that the resulting code has full total block length, which means that all 1-bit errors within that block length have separate

TABLE I

STANDARDIZED (STAND.) AND PRIMITIVE (PRIM.) GENERATOR POLYNOMIALS AND THEIR CORRESPONDING CRC SIGNATURES

$g(x)$ Utilized	Type	Predicted CRC-5 signatures	Actual CRC-5 signatures
$g_0(x) = x^5 + x^3 + 1$	Stand.	$(a_{15} + a_{12} + a_{11} + a_9 + a_8 + a_7 + a_5 + a_3)x^4 + (a_{12} + a_{11} + a_9 + a_8 + a_7 + a_6 + a_4 + a_2)x^3 + (a_{15} + a_{14} + a_{12} + a_{11} + a_{10} + a_8 + a_6 + a_1)x^2 + (a_{14} + a_{13} + a_{11} + a_{10} + a_9 + a_7 + a_5 + a_0)x + (a_{15} + a_{13} + a_{12} + a_{10} + a_9 + a_8 + a_6 + a_4)$	$(\gamma_{13} + \gamma_{12} + \gamma_{10} + \gamma_9 + \gamma_8 + \gamma_6 + \gamma_4)x^4 + (\gamma_{13} + \gamma_{12} + \gamma_{11} + \gamma_9 + \gamma_8 + \gamma_7 + \gamma_5 + \gamma_3)x^3 + (\gamma_{15} + \gamma_{13} + \gamma_{12} + \gamma_{11} + \gamma_9 + \gamma_7 + \gamma_2)x^2 + (\gamma_{15} + \gamma_{14} + \gamma_{12} + \gamma_{11} + \gamma_{10} + \gamma_8 + \gamma_6 + \gamma_1)x + (\gamma_{14} + \gamma_{13} + \gamma_{11} + \gamma_{10} + \gamma_9 + \gamma_7 + \gamma_5 + \gamma_0)$
$g_1(x) = x^5 + x^2 + 1$	Prim.	$(a_{14} + a_{13} + a_{12} + a_9 + a_8 + a_6 + a_3)x^4 + (a_{14} + a_{13} + a_{12} + a_{11} + a_8 + a_7 + a_5 + a_2)x^3 + (a_{15} + a_{14} + a_{13} + a_{12} + a_{11} + a_{10} + a_7 + a_6 + a_4 + a_1)x^2 + (a_{14} + a_{11} + a_{10} + a_8 + a_5 + a_0)x + (a_{15} + a_{14} + a_{13} + a_{10} + a_9 + a_7 + a_4)$	$(\gamma_{15} + \gamma_{14} + \gamma_{13} + \gamma_{10} + \gamma_9 + \gamma_7 + \gamma_4)x^4 + (\gamma_{15} + \gamma_{14} + \gamma_{13} + \gamma_{12} + \gamma_9 + \gamma_8 + \gamma_6 + \gamma_3)x^3 + (\gamma_{15} + \gamma_{14} + \gamma_{13} + \gamma_{12} + \gamma_{11} + \gamma_8 + \gamma_7 + \gamma_5 + \gamma_2)x^2 + (\gamma_{15} + \gamma_{12} + \gamma_{11} + \gamma_9 + \gamma_6 + \gamma_1)x + (\gamma_{15} + \gamma_{14} + \gamma_{11} + \gamma_{10} + \gamma_8 + \gamma_5 + \gamma_0)$
$g_2(x) = x^5 + x^4 + x^2 + x + 1$		$(a_{15} + a_{14} + a_{13} + a_{12} + a_{11} + a_9 + a_8 + a_5 + a_4 + a_3)x^4 + (a_{15} + a_{10} + a_9 + a_7 + a_5 + a_2)x^3 + (a_{14} + a_9 + a_8 + a_6 + a_4 + a_1)x^2 + (a_{12} + a_{11} + a_9 + a_7 + a_4 + a_0)x + (a_{15} + a_{14} + a_{13} + a_{12} + a_{10} + a_9 + a_6 + a_5 + a_4)$	$(\gamma_{15} + \gamma_{14} + \gamma_{13} + \gamma_{12} + \gamma_{10} + \gamma_9 + \gamma_6 + \gamma_5 + \gamma_4)x^4 + (\gamma_{11} + \gamma_{10} + \gamma_8 + \gamma_6 + \gamma_3)x^3 + (\gamma_{15} + \gamma_{10} + \gamma_9 + \gamma_7 + \gamma_5 + \gamma_2)x^2 + (\gamma_{13} + \gamma_{12} + \gamma_{10} + \gamma_8 + \gamma_5 + \gamma_1)x + (\gamma_{15} + \gamma_{14} + \gamma_{13} + \gamma_{11} + \gamma_{10} + \gamma_7 + \gamma_6 + \gamma_5 + \gamma_0)$
$g_3(x) = x^5 + x^4 + x^3 + x^2 + 1$		$(a_{15} + a_{14} + a_{13} + a_{12} + a_{11} + a_{10} + a_7 + a_4 + a_3)x^4 + (a_{15} + a_{14} + a_9 + a_7 + a_6 + a_4 + a_2)x^3 + (a_{15} + a_{12} + a_{11} + a_{10} + a_8 + a_7 + a_6 + a_5 + a_4 + a_1)x^2 + (a_{15} + a_{14} + a_{13} + a_{12} + a_9 + a_6 + a_5 + a_0)x + (a_{14} + a_{13} + a_{12} + a_{11} + a_8 + a_5 + a_4)$	$(\gamma_{15} + \gamma_{14} + \gamma_{13} + \gamma_{12} + \gamma_{11} + \gamma_8 + \gamma_5 + \gamma_4) \cdot x^4 + (\gamma_{15} + \gamma_{10} + \gamma_8 + \gamma_7 + \gamma_5 + \gamma_3)x^3 + (\gamma_{13} + \gamma_{12} + \gamma_{11} + \gamma_9 + \gamma_8 + \gamma_7 + \gamma_6 + \gamma_5 + \gamma_2)x^2 + (\gamma_{15} + \gamma_{14} + \gamma_{13} + \gamma_{10} + \gamma_7 + \gamma_6 + \gamma_1)x + (\gamma_{15} + \gamma_{14} + \gamma_{13} + \gamma_{12} + \gamma_9 + \gamma_6 + \gamma_5 + \gamma_0)$

remainders. Moreover, since the remainder is a linear function of the block, all 2-bit errors within that block length can be identified.

For the α module of the Luov's finite-field multipliers, $g_0(x) = x^5 + x^3 + 1$ is used as the standardized generator polynomial for CRC-5. To find its CRC signatures, this fixed polynomial is used as follows:

$$\begin{aligned} x^5 &\equiv x^3 + 1 \bmod g_0(x) \\ x^6 &\equiv x^4 + x \bmod g_0(x) \\ x^7 &\equiv x^5 + x^2 \equiv x^3 + x^2 + 1 \bmod g_0(x) \\ &\vdots \\ x^{15} &\equiv x^2 + 1 \bmod g_0(x). \end{aligned} \tag{2}$$

According to (1), we obtain $A(x) \cdot x = a_{15} \cdot x^{16} + a_{14} \cdot x^{15} + \cdots + a_1 \cdot x^2 + a_0 \cdot x$. Then, applying the irreducible polynomial $f(x) = x^{16} + x^{12} + x^3 + x + 1$, one obtains

$$\begin{aligned} A(x) \cdot x \equiv\ & a_{15}x^{12} + a_{15}x^3 + a_{15}x + a_{15} + a_{14}x^{15} \\ & + a_{13}x^{14} + a_{12}x^{13} + a_{11}x^{12} + a_{10}x^{11} + a_9x^{10} \\ & + a_8x^9 + a_7x^8 + a_6x^7 + a_5x^6 + a_4x^5 + a_3x^4 \\ & + a_2x^3 + a_1x^2 + a_1x \bmod f(x). \end{aligned} \tag{3}$$

To calculate the PCRC-5 for $GF(2^{16})$ in the α module ($PCRC5_{16}$), the generator polynomial is applied as

$$\begin{aligned} A(x) \cdot x \equiv\ & a_{15}(x^4 + x^3 + x^2 + x) + a_{15}x^3 + a_{15}x + a_{15} \\ & + a_{14}(x^2 + x) + a_{13}(x + 1) + a_{12}(x^4 + x^2 + 1) \\ & + a_{11}(x^4 + x^3 + x^2 + x) + a_{10}(x^3 + x^2 + x + 1) \\ & + a_9(x^4 + x + 1) + a_8(x^4 + x^3 + x^2 + 1) \\ & + a_7(x^4 + x^3 + x) + a_6(x^3 + x^2 + 1) + a_5(x^4 + x) \\ & + a_4(x^3 + 1) + a_3x^4 + a_2x^3 + a_1x^2 + a_0x \bmod g_0(x) \end{aligned}$$

or

$$\begin{aligned} \text{PCRC5}_{16} =\ & (a_{15} + a_{12} + a_{11} + a_9 + a_8 + a_7 + a_5 + a_3)x^4 \\ & + (a_{12} + a_{11} + a_9 + a_8 + a_7 + a_6 + a_4 + a_2)x^3 \\ & + (a_{15} + a_{14} + a_{12} + a_{11} + a_{10} + a_8 + a_6 + a_1)x^2 \\ & + (a_{14} + a_{13} + a_{11} + a_{10} + a_9 + a_7 + a_5 + a_0)x \\ & + (a_{15} + a_{13} + a_{12} + a_{10} + a_9 + a_8 + a_6 + a_4). \end{aligned} \tag{4}$$

To calculate the ACRC-5 for $GF(2^{16})$ in the α module ($ACRC5_{16}$), we rename the coefficients of (3): a_{14} as γ_{15}, …, a_0 as γ_1:

$$\begin{aligned} A(x) \cdot x \equiv\ & \gamma_{15}x^{15} + \gamma_{14}x^{14} + \gamma_{13}x^{13} + \gamma_{12}x^{12} \\ & + \gamma_{11}x^{11} + \gamma_{10}x^{10} + \gamma_9x^9 + \gamma_8x^8 + \gamma_7x^7 \\ & + \gamma_6x^6 + \gamma_5x^5 + \gamma_4x^4 + \gamma_3x^3 + \gamma_2x^2 + \gamma_1x^1 \\ & + \gamma_0 \bmod g_0(x) \end{aligned} \tag{5}$$

and the generator polynomial is applied as follows:

$$\begin{aligned} A(x) \cdot x \equiv\ & \gamma_{15}(x^2 + x) + \gamma_{14}(x + 1) + \gamma_{13}(x^4 + x^2 + 1) \\ & + \gamma_{12}(x^4 + x^3 + x^2 + x) + \gamma_{11}(x^3 + x^2 + x + 1) \\ & + \gamma_{10}(x^4 + x + 1) + \gamma_9(x^4 + x^3 + x^2 + 1) \\ & + \gamma_8(x^4 + x^3 + x) + \gamma_7(x^3 + x^2 + 1) + \gamma_6(x^4 + x) \\ & + \gamma_5(x^3 + 1) + \gamma_4x^4 + \gamma_3x^3 + \gamma_2x^2 \\ & + \gamma_1x^1 + \gamma_0 \bmod g_0(x) \end{aligned}$$

or

$$\begin{aligned} \text{ACRC5}_{16} =\ & (\gamma_{13} + \gamma_{12} + \gamma_{10} + \gamma_9 + \gamma_8 + \gamma_6 + \gamma_4)x^4 \\ & + (\gamma_{13} + \gamma_{12} + \gamma_{11} + \gamma_9 + \gamma_8 + \gamma_7 + \gamma_5 + \gamma_3)x^3 \\ & + (\gamma_{15} + \gamma_{13} + \gamma_{12} + \gamma_{11} + \gamma_9 + \gamma_7 + \gamma_2) \\ & \cdot x^2 + (\gamma_{15} + \gamma_{14} + \gamma_{12} + \gamma_{11} + \gamma_{10} + \gamma_8 + \gamma_6 + \gamma_1) \\ & \cdot x + (\gamma_{14} + \gamma_{13} + \gamma_{11} + \gamma_{10} + \gamma_9 + \gamma_7 + \gamma_5 + \gamma_0). \end{aligned} \tag{6}$$

The predicted output and the actual output are divided into five parity groups as shown in (4) and (6), respectively. These parity groups are XORed with each other to determine if there has been any fault, for example, flip of bits, during the α module operation. In total, each α module outputs five EFs. Fig. 2 shows the implementation of the α module with the proposed error-detection schemes. $A(x)$ is the input with the form $p(x) = a_{m-1}x^{m-1} + \cdots + a_1x + a_0$, which goes to two different modules that run in parallel. In the α module, (1) takes place. The output from the α module is divided into five groups in the ACRC module, which are denoted as x_a^1–x_a^5 in Fig. 2. Meanwhile, $A(x)$ is also being divided into five groups in the PCRC module, which are denoted as x_p^1–x_p^5. Once the two CRC modules are done, each group is XORed

TABLE II

OVERHEADS OF THE PROPOSED ERROR-DETECTION SCHEMES FOR THE FINITE-FIELD MULTIPLIERS USED IN THE LUOV ALGORITHM DURING THE POLYNOMIAL GENERATION ON XILINX FPGA FAMILY KINTEX ULTRASCALE+ FOR DEVICE XCKU5P-FFVD900-1-I

Architecture	Area (CLBs)	Delay (ns)	Power (mW) @50 MHz	Throughput (Gbps)	Efficiency ($Gbps/CLBs$)
Luov Multiplier	120	4.044	0.465	3.96	0.033
Luov Multiplier with standardized CRC-5 using $g_0(x)$	139 (15.83%)	4.194 (3.71%)	0.466 (≃0%)	3.81 (-3.79%)	0.027 (-18.18%)
Luov Multiplier with primitive CRC-5 using $g_1(x)$	134 (11.67%)	4.242 (4.90%)	0.466 (≃0%)	3.77 (-4.80%)	0.028 (-15.15%)
Luov Multiplier with primitive CRC-5 using $g_2(x)$	131 (9.17%)	4.252 (5.14%)	0.466 (≃0%)	3.76 (-5.05%)	0.028 (-15.15%)
Luov Multiplier with primitive CRC-5 using $g_3(x)$	142 (18.33%)	4.499 (11.25%)	0.466 (≃0%)	3.56 (-10.10%)	0.025 (-24.24%)

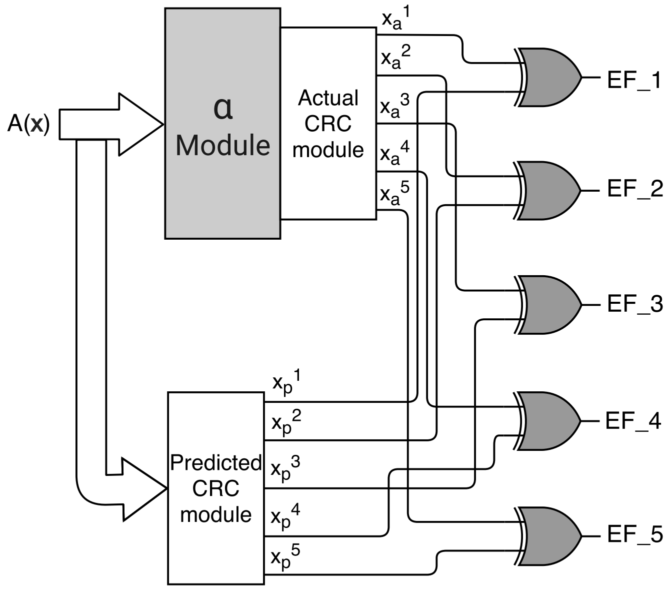

Fig. 2. Proposed error-detection constructions for α module.

with its respective one to produce five EFs, which are represented as EF_1–EF_5. As an example, to obtain EF_1, x_p^1 (or $a_{15} + a_{13} + a_{12} + a_{10} + a_9 + a_8 + a_6 + a_4$ for $g_0(x)$) is XORed with x_a^1 (or $\gamma_{14} + \gamma_{13} + \gamma_{11} + \gamma_{10} + \gamma_9 + \gamma_7 + \gamma_5 + \gamma_0$ for $g_0(x)$), which are calculated in (4) and (6), respectively. For our case study, the outputs are divided into five groups since we use CRC-5; however, if any other CRC-*n* is used, there will be *n* EFs and the actual and predicted outputs will be divided into *n* groups. In Table I, the CRC signatures for the different primitive polynomials are shown. We note that the choice of the utilized CRC can be tailored based on the reliability requirements and the overhead to be tolerated. In other words, for applications such as game consoles in which performance is critical (and power consumption is not because these are plugged in), one can increase the size of CRC. However, for deeply embedded systems such as implantable and wearable medical devices, smaller CRC is preferred.

IV. ERROR COVERAGE AND FPGA IMPLEMENTATIONS

Finite-field multiplication is a costly operation and requires large footprint. We implement Luov polynomial generation to show that the proposed error-detection schemes provide high error coverage with acceptable overhead. Such implementation produces a polynomial $p(x) = a_{m-1}x^{m-1} + \cdots + a_1x + a_0$, which requires $m-1$ finite-field multiplications and $m-1$ XOR operations. As pointed out before, each finite-field multiplication uses three different modules called α, sum, and $pass-thru$ modules. A total of $m-1$ α modules, $m-1$ sum modules, and m $pass-thru$ modules are needed to perform each finite-field multiplication. Moreover, a total of $m-1$ sum modules are needed to perform an XOR operation. For each architecture, the error coverage is calculated as $100 \cdot (1 - (1/2)^{\text{sign}})\%$, where $sign$ denotes the number of signatures.

Luov uses the finite-field $GF(2^{16})$, or $m = 16$. Implementing its polynomials in the form of $p(x) = a_{15}x^{15} + \cdots + a_1x + a_0$ requires 14 finite-field multiplications and 15 XOR operations. Since each finite-field multiplication uses $m-1$ α modules, $m-1$ sum modules, and m $pass-thru$ modules, 14×15 α modules, 14×15 sum modules, and 14×16 $pass-thru$ modules are needed. Moreover, a total of $14_{\text{multiplications}} \cdot (15_\alpha + 15_{\text{sum}} + 16_{\text{pass-thru}}) + 15_{\text{XOR}}$ or 659 signatures are implemented. The error coverage percentage for the generation of Luov's polynomial using the finite-field $GF(2^{16})$ is $100 \cdot (1 - (1/2)^{659})\%$. In Table II, we present the overhead of our error-detection architectures in terms of area-configurable logic blocks (CLBs), delay, power consumption (at the frequency of 50 MHz), throughput, and efficiency for the generation of polynomial $p(x)$, where $p(x) = a_{m-1}x^{m-1} + \cdots + a_1x + a_0$.

We utilize Xilinx FPGA family Kintex Ultrascale+ for device xcku5p-ffvd900-1-i, using Verilog as the hardware design entry and Vivado as the tool for the implementations. As shown in Table II, when CRC signatures are applied to the original architecture, with higher error coverage, they end up having higher overhead in terms of area, delay, and power, and lower overhead in terms of throughput and efficiency. CLBs, which are the main resources for implementing general-purpose combinational and sequential circuits, are read in the Vivado's place utilization report to obtain the area. To determine the delay, we use the Timing Constraints Wizard function in Vivado, setting a primary clock period constraint of 20 ns, which equals to a frequency of 50 MHz. We also report the total on-chip power, which is the power consumed internally within the FPGA and it is obtained by adding device static power and design power. Throughput is obtained by dividing the total number of output bits over the delay and efficiency is obtained by dividing throughput over area. As seen in this table, acceptable overheads are obtained with efficiency degradations of at most 19%. The error-detection architecture that uses the primitive generator polynomial $g_2(x)$ has the least amount of area overhead with 9.17%; however, the error-detection implementation using $g_0(x)$, or the standardized generator polynomial for CRC-5, performs the fastest, obtaining the least amount of delay overhead with 3.71%.

There has not been any prior work done on this type of error-detection methods for the Luov's finite-field multipliers to the best of our knowledge. For qualitative comparison to verify that the overheads incurred are acceptable, let us go over some case studies. Subramanian *et al.* [19] presented a signature-based fault diagnosis for cryptographic block Ciphers LED and HIGHT, obtaining a combined area and delay overhead of 21.9% and 31.9% for LED and HIGHT, respectively. Additionally, Mozaffari-Kermani *et al.* [6] have presented the fault diagnosis of Pomaranch cipher, obtaining a combined area and throughput overhead of 35.5%. The proposed schemes in this brief have combined area and delay overheads of less than 32% (worst case scenario). In [7], the worst case area overhead obtained by applying error-detection schemes of NTT architectures is 24%. The worst case area overhead of [8] and [9] is more than 33% with a performance degradation of more than 14% when fault-detection architectures are applied to stateless hash-based signatures. These and similar prior works on classical cryptography verify that the proposed error-detection architectures obtain similar overheads compared to other works on fault detection, achieving an acceptable overhead. These degradations are acceptable for providing error detection to the original architectures which lack such capability to thwart natural or malicious faults.

V. Conclusion

In this work, we have derived error-detection schemes for the finite-field multipliers used in postquantum cryptographic algorithms such as Luov, noting that the proposed error-detection schemes can be adapted to other applications and cryptographic algorithms whose building blocks need finite-field multiplications. The error-detection architectures proposed in this work are based on CRC-5 signatures and we have performed software implementations for the sake of verification. Additionally, we have explored and studied both primitive and standardized generator polynomials for CRC-5, comparing the complexity for each of them. We have embedded the proposed error-detection schemes into the original finite-field multipliers of the Luov's algorithm, obtaining high error coverage with acceptable overhead.

References

[1] J. L. Danger *et al.*, "On the performance and security of multiplication in $GF(2^N)$," *Cryptography*, vol. 2, no. 3, pp. 25–46, 2018.

[2] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "Reliable hardware architectures for the third-round SHA-3 finalist Grostl benchmarked on FPGA platform," in *Proc. DFT*, Oct. 2011, pp. 325–331.

[3] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "A low-cost S-box for the advanced encryption standard using normal basis," in *Proc. IEEE Int. Conf. Electro/Inf. Technol.*, Jun. 2009, pp. 52–55.

[4] M. Yasin, B. Mazumdar, S. S. Ali, and O. Sinanoglu, "Security analysis of logic encryption against the most effective side-channel attack: DPA," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFTS)*, Oct. 2015, pp. 97–102.

[5] M Mozaffari-Kermani, R. Azarderakhsh, A. Sarker, and A. Jalali, "Efficient and reliable error detection architectures of hash-counter-hash tweakable enciphering schemes," *ACM Trans. Embedded Comput. Syst.*, vol. 17, no. 2, pp. 54:1–54:19, May 2018.

[6] M. Mozaffari-Kermani, R. Azarderakhsh, and A. Aghaie, "Reliable and error detection architectures of Pomaranch for false-alarm-sensitive cryptographic applications," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 12, pp. 2804–2812, Dec. 2015.

[7] A. Sarker, M. Mozaffari-Kermani, and R. Azarderakhsh, "Hardware constructions for error detection of number-theoretic transform utilized in secure cryptographic architectures," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 3, pp. 738–741, Mar. 2019.

[8] M. Mozaffari-Kermani, R. Azarderakhsh, and A. Aghaie, "Fault detection architectures for post-quantum cryptographic stateless hash-based secure signatures benchmarked on ASIC," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 2, pp. 59:1–59:19, Dec. 2016.

[9] M. Mozaffari-Kermani and R. Azarderakhsh, "Reliable hash trees for post-quantum stateless cryptographic hash-based signatures," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFTS)*, Oct. 2015, pp. 103–108.

[10] M. M. Kermani and R. Azarderakhsh, "Reliable architecture-oblivious error detection schemes for secure cryptographic GCM structures," *IEEE Trans. Rel.*, vol. 68, no. 4, pp. 1347–1355, Dec. 2019.

[11] A. A. Kamal and A. M. Youssef, "Strengthening hardware implementations of NTRUEncrypt against fault analysis attacks," *J. Cryptograph. Eng.*, vol. 3, no. 4, pp. 227–240, Nov. 2013.

[12] A. Kipnis, J. Patarin, and L. Goubin, "Unbalanced oil and vinegar signature schemes," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.* Berlin, Germany: Springer, 1999, pp. 206–222.

[13] D. Moody, "Post-quantum cryptography: NIST's plan for the future," Tech. Rep., Feb. 2016. [Online]. Available: https://csrc.nist.gov/csrc/media/projects/post-quantum-cryptography/documents/pqcrypto-2016-presentation.pdf

[14] D. Moody, "Post-quantum cryptography: Round 2 submissions," Tech. Rep., Mar. 2019. [Online]. Available: https://csrc.nist.gov/CSRC/media/Presentations/Round-2-of-the-NIST-PQC-Competition-What-was-NIST/images-media/pqcrypto-may2019-moody.pdf

[15] D. J. Bernstein, "Post-quantum cryptography," in *Encyclopedia of Cryptography and Security*, H. C. A. van Tilborg and S. Jajodia, Eds. Boston, MA, USA: Springer, 2011, pp. 949–950, doi: 10.1007/978-1-4419-5906-5_386.

[16] A. Reyhani-Masoleh and M. A. Hasan, "Error detection in polynomial basis multipliers over binary extension fields," in *Proc. CHES*, 2002, pp. 515–528.

[17] *EPC Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz 960 MHz*, EPC Global, Brussels, Belgium, Version 1.0.23, 2008.

[18] T. V. Ramabadran and S. S. Gaitonde, "A tutorial on CRC computations," *IEEE Micro*, vol. 8, no. 4, pp. 62–75, Aug. 1988.

[19] S. Subramanian, M. Mozaffari-Kermani, R. Azarderakhsh, and M. Nojoumian, "Reliable hardware architectures for cryptographic block ciphers LED and HIGHT," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 10, pp. 1750–1758, Oct. 2017.